L13: Entry 1 of 3 File: USPT Dec 13, 2005

DOCUMENT-IDENTIFIER: US 6976027 B2
TITLE: Implementing geographical taxonomy within network-accessible service registries using spatial extensions

Brief Summary Text (2):
1. Field of the Invention

Brief Summary Text (3):
The present invention relates to computer programming, and deals more particularly with techniques for using geographical taxonomy data with spatial extensions (e.g., as extensions to an object-relational database) to facilitate programmatically locating information in network-accessible service registries.

Drawing Description Text (3):
FIG. 2 illustrates several data type structures, and relationships among them, in the UDDI data model of the prior art;

Drawing Description Text (4):
FIG. 3 is a Unified Modeling Language diagram showing how the data types in FIG. 2 are related;

Drawing Description Text (5):
FIGS. 4-7 provide schema definitions of the data types shown in FIG. 2, and

Drawing Description Text (6):
FIGS. 8, 9, and 11 provide schema definitions of supporting data types;

Drawing Description Text (8):
FIG. 12 provides a Unified Modeling Language diagram corresponding to the schema definitions in FIGS. 8, 9, and 11;

Detailed Description Text (4):
The UDDI registry specification defines several core data type structures, including "businessEntity", "businessService", "bindingTemplate", and "tModel". These data type structures, and relationships among them, are illustrated in FIG. 2. The primary role of the businessEntity data type 200 is to provide information about the entity publishing services in a registry, such as its name and contact information. A particular businessEntity instance may offer (i.e., publish) a number of business services in the registry, where each service is identified using an instance of the businessService data type 210. A businessService instance describes a service type in business terms, including a name for the service, a description for the service, and so forth.

Detailed Description Text (5):
Each businessService instance has an instance of bindingTemplate data type 220 for each way in which a service requester can access (i.e, invoke) the service. A bindingTemplate instance may contain a description, an access point definition which specifies how to call the service (e.g., as a Uniform Resource Identifier, or "URI"), etc. An instance of bindingTemplate may reference one or more instances of tModel data type 230. The primary role of a tModel is to point to a technical definition (i.e., interface specification) for a registered service. The tModels can be used as "technical fingerprints" to identify services. (In addition to

bindingTemplates, tModels may also be used in other data structures.)

Detailed Description Text (6):
FIG. 3 is a Unified Modeling Language ("UML") diagram showing how instances of the data types in FIG. 2 are related, and provides a more formal definition of the information just described textually. As is known in the art, UML diagrams provide a precise description of the relationship between various entities. (UML is a standard of the Object Management Group, and is described in "UML Toolkit", H. Eriksson, M. Penker, published by John Wiley and Sons, 1997.) Starting with the annotation tags across the bottom of FIG. 3, the purpose of each entity is documented. For example, this UML diagram indicates that each BusinessEntity 320 describes information about the company, and the tModel 350 is described as containing invocation details. The relationship between the entities are also shown. For example, the tModel 350 is associated with the BindingTemplate 340 such that there may be zero to many tModels associated with a single BindingTemplate. In turn, each BusinessService 330 has at least one, and perhaps many, BindingTemplates 340 associated with it.

Detailed Description Text (7):
The entities depicted in FIG. 3 will now be described in more detail. BusinessEntity 320 has a list of URLs that may be used to provide more information about the company (through "DiscoveryURL" instances; see element 300); a list of company contacts 310; and a list of the registered business services 330 offered by the company. For each BusinessService 330, a list of the available invocation descriptions, or BindingTemplates 340, is provided. The binding template may refer to a tModel 350 (or to multiple tModels), thereby implying certain specific and identifiable specifications that are provided. Thus, if an application needs to search for implementors of tModel "xyz", it can look for a binding template with a reference to this tModel.

Detailed Description Text (8):
Values for UDDI data structures are typically expressed using the Extensible Markup Language, or "XML". FIGS. 4-7 provide schema syntax which defines the format of the businessEntity, businessService, bindingTemplate, and tModel data types. (Refer to the UDDI specification for more detailed information about these data types.)

Detailed Description Text (9):
The businessEntity and tModel data types include a placeholder for elements named "identifierBag". See reference numbers 410 of FIG. 4 and 710 of FIG. 7. The schema for an identifierBag is shown in FIG. 8, and comprises one or more "keyedReference" elements, where each of these elements provides an identifier that may be used to identify the businessEntity or tModel. The schema for a keyedReference is shown in FIG. 9. As shown therein, a keyedReference comprises a tModelKey attribute, which is typically specified as a Universal Unique Identifier ("UUID") value; a keyName attribute, which is normally used for readability; and a keyValue attribute, which is used for specifying a value that is interpreted within a particular context. The context is identified by the tModelKey value. These attributes will be discussed in more detail below. (A "UUID" value is a 128-bit value which is guaranteed to be unique across all space and time.)

Detailed Description Text (10):
FIG. 10 provides example data values for an instance of the businessEntity data type. The sample businessEntity specification 1000 describes the IBM Corporation as a publisher of registered information (see reference number 1020 of FIG. 10). Three different identifiers 1031, 1032, and 1033 are provided in the identifierBag 1030 for this business entity. In this example, the business entity named "IBM Corporation" 1020 and having business key value "A#1" 1010 is therefore also identified using its Dun & Bradstreet number 1031, its U. S. tax identifier 1032, and its NAICS classification number 1033.

Detailed Description Text (12):
In addition to the identifierBag, the businessEntity and tModel data types also include a placeholder for elements named "categoryBag"; the businessService data type includes this placeholder as well. See reference numbers 420 of FIG. 4, 520 of FIG. 5, and 720 of FIG. 7. The categoryBag schema is provided in FIG. 11, and as shown therein, the structure of a categoryBag

instance is analogous to the structure of identifierBag (as shown in FIG. 8). FIG. 12 provides UML diagrams for the identifierBag and categoryBag structures, showing that identifierBag 1200 and categoryBag 1230 both contain a list of keyedReference instances 1210, where each keyedReference contains a reference to a tModel instance 1220.

Detailed Description Text (13):
An instance of categoryBag is used to provide the taxonomy values that categorize entries in the network-accessible registry. Similar to identifierBags, a keyedReference element within a categoryBag specifies a "keyValue" attribute that is interpreted in a particular namespace or context, where this context is identified by the corresponding tModelKey value. In the case of categoryBags, that context is a taxonomy used for categorization. Thus, entries in a categoryBag instance may be used to identify the categories with which a registered service (or a business entity or a tModel) is associated.

Detailed Description Text (17):
A businessService instance may also be categorized according to values from more than one taxonomy. Suppose, for example, that a completely different taxonomy also specifies values for computers and network-accessible package delivery scheduling software, perhaps using the value "14.15", where 14 corresponds to computers and 15 corresponds to package delivery scheduling software. If a keyedReference is not specified within a categoryBag according to this taxonomy, Acme's software will be overlooked on searches that are based on this taxonomy. The sample key values in this example have been chosen to emphasize that the values from the different taxonomies may be completely unrelated, yet still describe the same service.

Detailed Description Text (18):
As another example of using values in multiple taxonomies for a particular businessService instance, suppose that Acme Computer wants its scheduling software to be found if a search is performed for "all services based in Computer City, NY, U.S.A.", where Acme's facilities are located. If the GGC value for Computer City is "654321", then Acme can register its software under the GGC taxonomy by including an additional keyedReference element that identifies (via its tModelKey) the GGC taxonomy, where this keyedReference element specifies the value "654321" for the keyValue attribute. (Note that when multiple keyedReference elements are specified, they may be in a single categoryBag or in more than one categoryBag.)

Detailed Description Text (21):
Geographic information systems are known in the art, and store geographic or cartographic (i.e., map-oriented) data. Systems are also known in the art for using relational databases to process (e.g., store and access) this type of geographic data. When a relational database is adapted for use with geographic information system ("GIS") data, the database is often referred to as "spatially-enabled".

Detailed Description Text (23):
Geographic data may describe the physical location or area of a place or thing, or even the location of a person. When geographic data is stored in a spatially-enabled database, it is stored using a geometric model in which locations/areas are expressed in terms of geometric shapes or objects. The geometric data stored according to this model may also be referred to as "spatial data". In addition to locations or areas of geographic objects, spatial data may also represent relationships among objects, as well as measurements or distances pertaining to objects. As an example of relationships among objects, spatial data may be used to determine whether a geometric shape corresponding to the location of a particular bridge intersects a geometric shape corresponding to the location of a river (thus determining whether the bridge crosses the river). As an example of using spatial data for measurements or distances, the length of a road passing through a particular county could be determined using the geometric object representing the road and a geometric object which specifies the boundaries of the county.

Detailed Description Text (24):
Spatial data values are expressed in terms of "geometry" or "geometric" data types. Thus, the location of a landmark might be expressed as a point having (x,y) coordinates, and the perimeter of a lake might be defined using a polygon. Typical spatially-enabled database

systems support a set of basic geometry data types and a set of more complex geometry data types, where the basic types comprise points, line strings, and polygons, and the complex types comprise collections of points, collections of line strings, and collections of polygons.

Detailed Description Text (25):
A common geometric model used by spatially-enabled database systems is shown in FIG. 14. As shown therein, the model is structured as a hierarchy or tree 1400 having geometry 1405 as its root, and having a number of subclasses. Point 1410, linestring 1420, and polygon 1430 represent the basic geometry data types. In this model 1400, linestring 1420 is a subclass of curve 1415, and polygon 1430 is a subclass of surface 1425. Geometry collection class 1435 is the root of a subtree representing the more complex geometric data types, and each subclass thereof is a homogeneous collection. Multipolygon 1445, multistring 1455, and multipoint 1460 represent the collections of polygons, line strings, and points, respectively. Multipolygon 1445 is a subclass of multisurface 1440 in this model, and multistring 1455 is a subclass of multicurve 1450. Only the classes which are leaves of this tree 1400 (i.e., 1410, 1420, 1430, 1445, 1455, and 1460) are instantiable in typical spatially-enabled database systems; the other nodes correspond to abstract classes. (Each of these entities is an actual data type.)

Detailed Description Text (26):
Referring now to the basic data types in particular, geometric data according to the model 1400 of FIG. 14 may be expressed in terms of a single point having (x,y) coordinates, or may be described as a line string or a polygon. A line string may be considered as one or more line segments which are joined together, and is defined using an ordered collection of (x,y) coordinates (i.e. points) that correspond to the endpoints of the connected segments. A polygon is defined using an ordered collection of points at which a plurality of line segments end, where those line segments join to form a boundary of an area.

Detailed Description Text (28):
Once spatial information has been stored in a database, the database can be queried to obtain many different types of information, such as the distance between two cities, whether a national park is wholly within a particular state, and so forth.

Detailed Description Text (29):
As one example of a spatially-enabled database, a feature known as "Spatial Extender" can be added to IBM's DB2.RTM. relational database product to provide GIS support. Spatial Extender provides support for the geometric data types shown in FIG. 14, and provides a number of built-in functions for operating on those data types. When using Spatial Extender, spatial data can be stored in columns of spatially-enabled database tables by importing the data or deriving it. The import process uses one of several input formats (such as text or binary data, the details of which are not pertinent to the present invention), and processes that data using built-in functions to convert it to geometric data. Spatial data may be derived either by operating on existing geometric data (for example, by defining a new polygon as a function of an existing polygon) or by using a process known as "geocoding". A geocoder is provided with Spatial Extender that takes as input an address in the United States and derives a geometric point representation. Other geocoders can be substituted to provide other types of conversions.

Detailed Description Text (32):
Alternatively, a taxonomy that supports references to geospatially-enabled databases might be registered, in which case the standard UDDI types taxonomy may then be used with embodiments of the present invention.

Detailed Description Text (33):
As an example of categorizing a registered service according to the present invention, suppose that a geospatially-enabled database contains an entry specified using the polygon 1430 data type, where this polygon defines the boundary of Computer City, NY. Or, the database might contain an entry specified using the point 1410 data type, where this point represents the town of Computer City. A keyvalue attribute that resolves to this polygon or point, as appropriate, can then be used to associate Acme's scheduling software with Computer City. A search for registered services from this geography can then be conducted.

Detailed Description Text (34):
In addition to using points or polygons to define the boundary of a city, additional or different types of geographic entities can be associated with a registered service using these or other geometric data types. For example, a search might be conducted to locate all registered package delivery services operating within the state of New York, and when the services have specified a keyedReference whose value resolves to the state in which they operate, the set of services within New York can be efficiently determined. Other functions which are available through spatially-enabled databases can then be used as well, for example to efficiently compute the distance between a customer location and the available drop-ship locations for a customer's order.

Detailed Description Text (35):
For the standard UDDI types taxonomy, values of the keyName attribute within a keyedReference element serve only descriptive purposes. However, when using the general.sub.-- keywords taxonomy, the keyName attribute takes on additional meaning. In particular, the keyName identifies the set of values from which the value of the keyvalue attribute is taken. Therefore, according to the UDDI specification, a keyName value is preferably specified as a Uniform Resource Name ("URN") when using the general.sub.-- keywords taxonomy, and care should be taken to avoid name collisions.

Detailed Description Text (39):
The first categoryBag 1510 includes a first keyedReference element categorizing this service with reference to the UNSPSC taxonomy, and a hypothetical sub-category "Software", in which a hypothetical category value of "01.02.03.04.05" is associated with this registered service. (The value of the tModelKey is a UUID that points to the tModel for the UNSPSC taxonomy.) This is an example of a prior art categorization. A second keyedReference element in this categoryBag 1510 specifies its value with reference to a geographical taxonomy, according to the present invention. In this example, the geographical taxonomy is identified by its URN "categorizeComputers.sample:taxonomies:Zone7". In this taxonomy, the service has a categorization value of "1235". So, for example, the taxonomy " . . . Zone7" might be used for categorizing locations within the United States Department of Agriculture ("USDA") plant hardiness zone number 7, where the value "1235" might identify a particular subdivision of that zone. According to the present invention, this taxonomy name and its associated value then point to a geometric data type in a spatially-enabled database, where the geometric data type preferably provides a description of an associated location. (USDA plant hardiness zones identify irregularly-shaped bands across the United States.)

Detailed Description Text (40):
The second categoryBag 1520 includes a categorization value of "1231" in the NAICS taxonomy, which is an example of a prior art categorization of this service. This categoryBag 1520 also includes a categorization of the type disclosed herein, where the value "1234" is associated with this service according to the classification scheme identified using the URN "categorizeComputers.sample:taxonomies:Area7". This " . . . Area7" taxonomy may specify some arbitrary geographic classification system, and the value "1234" within this system is associated with the registered service by inclusion of this keyedReference element.

Detailed Description Text (45):
An advantage of using the geographical taxonomy data as disclosed herein is that multiple names for a particular geometric data type can be used within the keyedReference elements, and these can be resolved automatically. Thus, fewer entries can be used in the categoryBags. For example, the publisher of service 1500 only needs to include a keyedReference element specifying the keyValue of 1235 within the " . . . Zone7" taxonomy, or the keyValue of 1234 within the " . . . Area7" taxonomy, because both of these identify the same row in the spatially-enabled database. A search using either approach will therefore locate registered service 1500.

Detailed Description Text (49):
Using the spatial extensions of the present invention, more granular categorization can be provided using less data, and searches can be provided more efficiently and more effectively. The disclosed techniques work within defined and accepted UDDI access specifications, and

extend existing categorization techniques in an advantageous manner. In this manner, creation of unique, ill-defined taxonomies can be avoided. Existing relational database technology can be leveraged for data normalization and data management. The spatial extensions, geometric data types, grid indexing functions, user-defined functions, and built-in procedures of the database system can also be used. In this manner, operations on the stored data can use optimized built-in functions of the database system, rather than requiring an applications programmer to provide complex code in his/her application. As a result, programmer efficiency is increased and code complexity is reduced, thereby leading to decreased program development and support costs. Furthermore, use of the optimized built-in database functions for interacting with the stored data will typically increase the efficiency of application programs and search utilities.

Other Reference Publication (2):
UDDI tModels "Classification Schemes, Taxonomies, Identifier Systems, and Relationships" Nov. 15, 2001, http://uddi.org/taxonomies/UDDI.sub.-- Taxonomy.sub.-- tModels.htm, pp. 1-21.

Previous Doc Next Doc Go to Doc#

L14: Entry 1 of 1 File: USPT Nov 27, 2001

DOCUMENT-IDENTIFIER: US 6324533 B1
TITLE: Integrated database and data-mining system

Abstract Text (1):
A method and apparatus for mining data relationships from an integrated database and data-mining system are disclosed. A set of frequent 1-itemsets is generated using a group-by query on data transactions. From these frequent 1-itemsets and the transactions, frequent 2-itemsets are determined. A candidate set of (n+2)-itemsets are generated from the frequent 2-itemsets, where n=1. Frequent (n+2)-itemsets are determined from candidate set and the transaction table using a query operation. The candidate set and frequent (n+2)-itemset are generated for (n+1) until the candidate set is empty. Rules are then extracted from the union of the determined frequent itemsets.

Brief Summary Text (1):
FIELD OF THE INVENTION

Brief Summary Text (2):
The present invention generally relates to data processing, and more particularly, to a method and system for efficiently mining data relationships from an integrated database and data-mining system.

Brief Summary Text (5):
Recently, researchers have started to focus on issues related to integrating mining with databases, such as proposals to extend the SQL language to support mining operators. For instance, the query language DMQL proposed by Han et al. extends SQL with a collection of operators for mining characteristic rules, discriminant rules, classification rules, association rules, etc. In the paper "Discovery Board Application Programming Interface and Query Language for Database Mining," Proc. of the 2nd Int'l Conference on Knowledge Discovery and Data Mining, Oregon, August 1996, Imielinski et al. extend M-SQL, which is an extension of the SQL language with a special unified operator to generate and query a whole set of propositional rules. Another example is the mine rule operator proposed by Meo et al. for a generalized version of the association rule discovery problem, described in "A New SQL Like Operator For Mining Association Rules," Proc. of the 22nd Int'l Conference on Very Large Databases, India, September 1996. Tsur et al. also proposed "query flocks" for data mining using a generate-and-test model, as described in "Query Flocks: A Generalization of Association Rule Mining," available on the World Wide Web at http://db.stanford.edu/ullman/pub/flocks.ps, October 1997.

Brief Summary Text (9):
It is an object of the present invention to provide an efficient method for extracting data relationships from an integrated database and data-mining system.

Brief Summary Text (10):
Another object of the present invention is to provide a method for mining data relationships from the integrated mining system in the form of queries to SQL engines, and with k-way join, three-way join, subqueries, and group-by operations for counting the itemset support.

Brief Summary Text (11):

Still another object of the present invention is to provide a method for mining data relationships from the integrated mining system in the form of queries to SQL engines enhanced with object-relational extensions (SQL-OR), such as user-defined functions (UDFs) and table functions.

Brief Summary Text (12):
The present invention achieves the foregoing and other objects by providing a method for identifying rules from an integrated database and data-mining system, where the system includes a database in the form of a table of transactions and a query engine. The method includes the steps of: a) performing a group-by query on the transaction table to generate a set of frequent 1-itemsets; b) determining frequent 2-itemsets from the frequent 1-itemsets and the transaction table; c) generating a candidate set of (n+2)-itemsets from the frequent (n+1)-itemsets, where n=1; d) determining frequent (n+2)-itemsets from the candidate set of (n+2)-itemsets and the transaction table using a query; e) repeating steps (c) and (d) with n=n+1 until the candidate set is empty; and f) generating rules from the union of the determined frequent itemsets.

Brief Summary Text (13):
The group-by query preferably includes the steps of counting the number of transactions that contain each item and selecting the items that have a support above a user-specified threshold in determining the frequent 1-itemsets. The support of an item is the number of transactions that contain the item. In determining the frequent 2-itemsets, each 1-itemset is joined with itself and two copies of the transaction table using join predicates. The joining results for a pair of items are grouped together in counting the support of the items in the pair. All 2-itemsets that have a support below a specified threshold are removed from the set of 2-itemsets, resulting in the frequent 2-itemsets.

Brief Summary Text (14):
Preferably, the candidate (n+2)-itemsets are generated using an (n+2)-way join operation. Also, the frequent (n+2)-itemsets are determined using an (n+3)-way join query on the candidate itemsets and (n+2) copies of the transaction table. Alternatively, the frequent (n+2)-itemsets are determined using cascaded subqueries by: a) selecting distinct first items in the candidate itemsets using a subquery; b) joining the distinct first items with the transaction table; c) cascading (n+1) subqueries where each j-th subquery is a three-way join of the result of the last subquery, distinct j items from the candidate itemsets, and the transaction table; and d) using the results of the last subqueries to determine which of the (n+2)-itemsets are frequent.

Brief Summary Text (15):
In generating rules from the union of the frequent itemsets, all items from the frequent itemsets are first put into a table F. A set of candidate rules is created from the table Fusing a table function. These candidate rules are joined with the table F, and filtered to remove those that do not meet a confidence criteria.

Brief Summary Text (16):
In the case where the mining engine is an object-relational engine, the generation of 2-itemsets and (n+1)-itemsets preferably includes: a) selecting from Vertical, GatherCount, and GatherJoin a method that has the best cost for generating the (n+1)-itemsets; b) if the Vertical method is selected, then transforming the transaction table into a vertical format if it is not already transformed; and c) generating the frequent (n+1)-itemsets using the selected method.

Drawing Description Text (4):
FIG. 3 is a flowchart representing the general operation of a basic method for mining data relationships in an integrated database and data-mining system, in accordance with the invention.

Drawing Description Text (7):
FIG. 6 illustrates the rule generation query in which a table function is used to generate all possible rules from a frequent itemset.

Drawing Description Text (8):
FIG. 7 is a flowchart showing further details of the process of determining frequent 2-itemsets (from step 31 of FIG. 3).

Drawing Description Text (10):
FIG. 9 is a flowchart showing further details for the step of determining frequent (n+2)-itemsets (step 33 of FIG. 3) using a subqueries-based approach.

Detailed Description Text (2):
The invention will be described primarily in terms of methods for mining relationships among data items from an integrated system of a database and a data-mining engine. However, persons skilled in the art will recognize that a computing system, which includes suitable programming means for operating in accordance with the methods to be disclosed, also falls within the spirit and scope of the invention. In addition, the invention may also be embodied in a computer program product, such as a diskette, for use with a suitable data processing system. Programming means is also provided for directing the data processing system to execute the steps of the methods of the invention, as embodied in the program product.

Detailed Description Text (10):
Given a set of transactions, where each transaction is a set of items, an association rule is an expression X.fwdarw.Y, where X and Y are sets of items. See, for example, "Mining Association Rules Between Sets Of Items In Large Databases," R. Agrawal et al., Proc. of the ACM SIGMOD Conference on Management of Data, pp. 207-216, Washington, D.C., May 1993. The intuitive meaning of such a rule is that transactions in the database which contain the items in X tend to also contain the items in Y. An example of such a rule might be that "30% of transactions that contain beer also contain diapers; 2% of all transactions contain both these items". Here 30% is called the CONFIDENCE of the rule, and 2% the SUPPORT of the rule. The problem of mining association rules is to find all rules that satisfy a user-specified minimum support and minimum confidence. This problem can be decomposed into two subproblems:

Detailed Description Text (11):
Find all combinations of items whose support is greater than minimum support. Call those combinations frequent itemsets; and

Detailed Description Text (12):
Use the frequent itemsets to generate the desired rules. The idea is that if, say, ABCD and AB are frequent itemsets, then it can be determined whether the rule AB.fwdarw.CD holds by computing the ratio r=support(ABCD)/support(AB). The rule holds only if r>minimum confidence. Note that the rule will have minimum support because ABCD is frequent.

Detailed Description Text (14):
The basic Apriori method for finding frequent itemsets makes multiple passes over the data. In the k-th pass it finds all itemsets having k items called the k-itemsets. Each pass consists of two phases. Let F.sub.k represent the set of frequent k-itemsets, and C.sub.k the set of candidate k-itemsets (potentially frequent itemsets). First, in the candidate generation phase, the set of all frequent (k-1)-itemsets, F.sub.k-1, found in the (k-1)-th pass, is used to generate the candidate itemsets C.sub.k. The candidate generation procedure ensures that C.sub.k is a superset of the set of all frequent k-itemsets. The algorithm builds a specialized hash-tree data structure in memory out of C.sub.k. Data is then scanned in the support counting phase. For each transaction, the algorithm determines which of the candidates in C.sub.k are contained in the transaction using the hash-tree data structure and increments their support count. At the end of the pass, C.sub.k is examined to determine which of the candidates are frequent, yielding F.sub.k. The algorithm terminates when F.sub.k or C.sub.k+1 becomes empty.

Detailed Description Text (16):
Input format: The transaction table T normally has two column attributes: transaction identifier (tid) and item identifier (item). The number of items per transaction is variable and unknown during table creation time. Thus, alternative schemes may not be convenient. In particular, it is not practical to assume that all items in a transaction appear as different columns of a single tuple because often the number of items per transaction can be more than

the maximum number of columns that the database supports. For instance, for one of our real-life datasets the maximum number of items per transaction is 872 and for another it is 700. In contrast, the corresponding average number of items per transaction is only 9.6 and 4.4 respectively.

Detailed Description Text (17):
Output format: The output is a collection of rules of varying length. The maximum length of these rules is much smaller than the number of items and is rarely more than a dozen. Therefore, a rule is represented as a tuple in a fixed-width table where the extra column values are set to NULL to accommodate rules involving smaller itemsets. The schema of a rule is (item.sub.1, . . . , item.sub.k, len, rulem, confidence, support) where k is the size of the largest frequent itemset. The len attribute gives the length of the rule and the rulem attribute gives the position of the.fwdarw.in the rule.

Detailed Description Text (20):
FIG. 2 schematically shows the configuration of an integrated database and data-mining system. The mining operation is expressed in some extension of SQL or a graphical language, which are input to preprocessor 21. This preprocessor generates appropriate SQL translations for the mining operation. For example, these SQL translations may be those that are executed by a SQL-92 relational engine 22, or by an object-relational engine 23 having the newer object-relational capabilities for an SQL (SQL-OR). The SQL-92 engine is described by J. Melton et al. in the book entitled "Understanding The New SQL: A Complete Guide," Morgan and Kauffman, 1992. Object-relational extensions are described by K. Kulkarni et al. in "Object Oriented Extensions in SQL3: A Status Report," SIGMOD Record, 1994. It is assumed that blobs, user-defined functions, and table functions are available in the object-relational engine. The mining results might be output to a depository 24.

Detailed Description Text (21):
FIG. 3 is a flow chart showing the general operation of a method for mining data in an integrated database and data-mining system. Start with step 30, a group-by query is performed on the data transactions to generate a set of frequent 1-itemsets. One-itemsets are those having exactly one item each, while an itemset is frequent if the number of transactions containing it is at least at a specified number. At step 31, frequent 2-itemsets are determined from the frequent 1-itemsets and the transaction table. A candidate set of (n+2)-itemsets is next generated in step 32 from the frequent (n+1)-itemsets, where n=1. At step 33, frequent (n+2)-itemsets are generated from the candidate set of (n+2)-itemsets and the transaction table using a query. Steps 32 and 33 are repeated for the next value of n until the candidate set is empty, as shown by step 35. When the candidate set is empty, rules are generated from the union of the determined frequent itemsets in step 36. Further details of the steps in FIG. 3 are now described.

Detailed Description Text (23):
The basic process for finding frequent itemsets proceeds as follows. In each pass k of the algorithm, a candidate itemset set C.sub.k is generated from frequent itemsets F.sub.k-1 of the previous pass. Given the set of all frequent (k-1)-itemsets, F.sub.k-1, the candidate generation procedure returns a superset of the set of all frequent k-itemsets. The items in an itemset are assumed to be lexicographically ordered. Since all subsets of a frequent itemset are also frequent, the candidate itemset set C.sub.k can be generated from F.sub.k-1 as follows:

Detailed Description Text (24):
First, in the join step, a superset of the candidate itemsets C.sub.k is generated by joining F.sub.k-1 with itself, as shown in the following pseudo code:

Detailed Description Text (31):
Next, in the prune step, all itemsets c.epsilon. C.sub.k, where some (k-1)-subset of c is not in F.sub.k-1, are deleted. Continuing with the example above, the prune step will delete the itemset {1 3 4 5} because the subset {1 4 5} is not in F.sub.3. C.sub.4 is then left with only {1 2 3 4}. The prune step can be done in the same SQL statement as the join step above by writing it as a k-way join as shown in FIG. 4. A k-way join is used since for any k-itemset

there are k subsets of length (k-1) for which we need to check in F.sub.k-1 for membership. The join predicates on I.sub.1 and I.sub.2 remain the same. After the join between I.sub.1 and I.sub.2 we get a k itemset consisting of (I.sub.1.item.sub.2, I.sub.1.item.sub.1, . . . I.sub.1.item.sub.k-1, I.sub.2.item.sub.k-1) as shown above. For this itemset, two of its (k-1)-length subsets are already known to be frequent since it was generated from two itemsets in F.sub.k-. The remaining k-2 subsets are checked using additional joins. The predicates for these joins are enumerated by skipping one item at a time from the k-itemset as follows: item, is skipped and the subset (I.sub.1.item.sub.2, . . . , I.sub.1.item.sub.k-1, I.sub.2.item.sub.k-1) is checked to see if it belongs to F.sub.k-1 as shown by the join with 13 in the figure. In general, for a join with I.sub.r item r-2 is skipped. FIG. 5 shows an example of the candidate generation for k=4.

Detailed Description Text (32):
A primary index is constructed on (item.sub.1, . . . , item.sub.k-1,) of F.sub.k-1 to efficiently process these k-way joins using index probes. Note that sometimes it may not be necessary to materialize C.sub.k before the counting phase. Instead, the candidate generation can be pipelined with the subsequent SQL queries used for support counting.

Detailed Description Text (33):
2. Counting support to find frequent itemsets

Detailed Description Text (34):
The candidate itemsets C.sub.k and data table T are used for counting the support of the itemsets in C.sub.k. Two SQL categories may be used:

Detailed Description Text (36):
2. SQL object-relational extensions (SQL-OR): Five preferred embodiments based on SQL-OR extensions like user-defined functions (UDFs), binary large objects (BLOBs), and table functions, will be described below in section 5. Table functions are virtual tables associated with a user defined function which generate tuples on the fly. See, for example, "Extensions To Starburst: Objects, Types, Functions, and Rules," G. Lohman et al., Communications of the ACM, 34(10), October 1991. Like normal physical tables, they have pre-defined schemas. The function associated with a table function can be implemented as any other UDF. Thus, table functions can be viewed as UDFs that return a collection of tuples instead of scalar values.

Detailed Description Text (38):
In order to generate rules having minimum confidence (minconf), all non-empty proper subsets of every frequent itemset/are determined. For each subset m, a confidence of the rule m.fwdarw.(I-m) is derived. This rule is then output if it satisfies at least minconf.

Detailed Description Text (39):
In the support counting phase, the frequent itemsets of size k are stored in table F.sub.k. Before the rule generation phase, all the frequent itemsets are merged into a single table F. The schema of F consists of k+2 attributes (item.sub.1, . . . , item.sub.k, support, len), where k is the size of the largest frequent itemset and len is the length of the itemset.

Detailed Description Text (40):
A table function, GenRules, is used to generate all possible rules from a frequent itemset. The input to the function is a frequent itemset. For each itemset, it outputs tuples corresponding to rules with all non-empty proper subsets of the itemset in the consequent. The table function outputs tuples with k+3 attributes, T_item.sub.1, . . . , T_item.sub.k, T_support, T_ten, T_rulem). The output is joined with F to find the support of the antecedent and the confidence of the rule is calculated by taking the ratio of the support values. FIG. 6 illustrates the rule generation query process, while the following pseudo-code shows a typical implementation of this step.

Detailed Description Text (41):
insert into R select T_item.sub.1, . . . , T_item.sub.k, t.sub.1. support, T_len, T_rulem, t.sub.1. support/f.sub.2. support

Detailed Description Text (42):
from F f.sub.1, table(GenRules(f.sub.1. item.sub.1, . . . , f.sub.1. item.sub.k, f.sub.1. len, f.sub.1. support)) as t.sub.1, F f.sub.2 where (t.sub.1. T_item.sub.1 =f.sub.2. item.sub.1 or t.sub.1.T_rulem>1)

Detailed Description Text (43):
AND (t.sub.1.T_item.sub.k =f.sub.2.item.sub.k or t.sub.1.T_rulem>k)

Detailed Description Text (46):
The above rule generation can also be done using SQL-92 and without the table functions. The rules are generated in a level-wise manner in which rules are generated with consequents of size k in each level k. Further, for any frequent itemset, if a rule with consequent c holds then so do rules with consequents that are subsets of c. With this property, the rules in level k are formed using rules with (k-1) long consequents found in the previous level, as in the previously described candidate generation.

Detailed Description Text (50):
In each pass k, the candidate itemsets C.sub.k is joined with k transaction tables T. This is followed by a group-by operation on the itemsets as shown by the following pseudo-code.

Detailed Description Text (51):
insert into F.sub.k select item.sub.1, . . . , item.sub.k, count(*)

Detailed Description Text (57):
group by item.sub.1, item.sub.2 . . . item.sub.k

Detailed Description Text (59):
For the second pass, a special optimization is used where instead of materializing C.sub.2, it is replaced with the 2-way joins between the two copies of F.sub.1. This saves the cost of materializing C.sub.2 and also provides early filtering of the T based on F.sub.1 instead of the larger C.sub.2. In contrast, for other passes corresponding to k>2, C.sub.k could be smaller than F.sub.k-1 because of the prune step. FIG. 7 is a flowchart showing further details of the process of determining frequent 2-itemsets (from step 31 of FIG. 3), while FIG. 8 shows a tree diagram of the query. This tree diagram is not to be confused with the plan trees which could look quite different. This SQL computation, when merged with the candidate generation step, can be used as a possible mechanism to implement query flocks.

Detailed Description Text (61):
The above approach requires (k+1)-way joins in the k-th pass. The cardinality of joins can be reduced to 3 using the following approach. Each candidate itemset C.sub.k, in addition to attributes (item.sub.1, . . . , item.sub.k) has three new attributes (oid, id.sub.1, id.sub.2). oid is a unique identifier associated with each itemset, and id.sub.1 and id.sub.2 are oids of the two itemsets in F.sub.k-1 from which the itemset in C.sub.k was generated. In addition, in the k-th pass a new copy of the data table T.sub.k with attributes (tid, oid) is generated that keeps for each tid the oid of each itemset in C.sub.k that it supported. For support counting, T.sub.k is first generated from T.sub.k-1 and C.sub.k. A group-by operation is performed on T.sub.k to find F.sub.k as follows:

Detailed Description Text (65):
insert into F.sub.k select oid, item.sub.1, . . . item.sub.k, cnt

Detailed Description Text (71):
This approach makes use of common prefixes between the itemsets in C.sub.k to reduce the amount of work done during support counting. The support counting phase is broken into a cascade of k subqueries. The l-th subquery Q.sub.l finds all tids that match the distinct itemsets formed by the first l columns of C.sub.k (call it dl). The output of Q.sub.l is joined with T and d.sub.l+1 (the distinct itemsets formed by the first l+1 columns of C.sub.k) to get Q.sub.l+1. The final output is obtained by doing a group-by on the k items to count support as above. For k=2, the same query as in the k-join approach is used. FIG. 9 is a flowchart showing further details for the step of determining frequent (n+2)-itemsets, while the following pseudo-code

represents a typical implementation of this step.

Detailed Description Text (72):
insert into F.sub.k select item.sub.1, . . . item.sub.k, count(*)

Detailed Description Text (74):
group by item.sub.1, item.sub.2 . . . item.sub.k

Detailed Description Text (77):
select item.sub.1 . . . item.sub.1, tid

Detailed Description Text (79):
(select distinct item.sub.1 . . . item, from C.sub.k) as d.sub.1

Detailed Description Text (86):
This approach avoids the multi-way joins used in the previous approaches, by joining T and k based on whether the "item" of a (tid, item) pair of T is equal to any of the k items of C.sub.k. Then, do a group by on (item.sub.1, . . . , item.sub.k, tid) filtering tuples with count equal to k. This gives all (itemset, tid) pairs such that the tid supports the itemset. Finally, as in the previous approaches, do a group-by on the itemset (item.sub.1, . . . , item.sub.k) filtering tuples that meet the support condition.

Detailed Description Text (87):
insert into F.sub.k select item.sub.1, . . . , item.sub.k, count(*)

Detailed Description Text (88):
from (select item.sub.1, . . . , item.sub.k, count(*)

Detailed Description Text (90):
where item=C.sub.k.item.sub.1 or

Detailed Description Text (91):
item=C.sub.k.item.sub.k

Detailed Description Text (92):
group by item.sub.1, . . . , item.sub.k, tid

Detailed Description Text (94):
group by item.sub.1, . . . , item.sub.k

Detailed Description Text (96):
5. Support Counting Using SQL With Object-Relational Extensions

Detailed Description Text (97):
The performance of an integrated database and data-mining system may be further improved using additional object-relational extensions to SQL for counting the itemset support. Such a method, called GatherJoin, and its variants, GatherCount, GatherPrune, and Horizontal, are now described.

Detailed Description Text (99):
This approach is based on the use of table functions described above. It generates all possible k-item combinations of items contained in a transaction, joins them with the candidate table C.sub.k, and counts the support of the itemsets by grouping the join result. Two table functions, Gather and Comb-K, are used. The data table T is scanned in the (tid, item) order and passed to the table function Gather. This table function collects all the items of a transaction (in other words, items of all tuples of T with the same tid) in memory and outputs a record for each transaction. Each such record consists of two attributes, the tid and item-list which is a collection of all its items in a VARCHAR or a BLOB. The output of Gather is passed to another table function Comb-K which returns all k-item combinations formed out of the items of a transaction. A record output by Comb-K has k attributes T_itm.sub.1, . . . ,

T_itm.sub.k, which can be directly used to probe into the C.sub.k table. An index is constructed on all the items of C.sub.k to make the probe efficient. FIG. 10 illustrates the SQL queries for the GatherJoin approach.

Detailed Description Text (100):
This approach is analogous to the K-way Join approach where the k-way self join of T is replaced with the table functions Gather and Comb-K. It is possible to merge these functions together as a single table function GatherComb-K. The Gather function is not required when the data is already in a horizontal format where each tid is followed by a collection of all its items. The pseudo-code below illustrate a typical implementation of GatherJoin approach for counting support.

Detailed Description Text (101):
insert into F.sub.k select item.sub.1, . . . , item.sub.k, count(*)

Detailed Description Text (110):
Note that for k=2, the 2-candidate set C.sub.2 is simply a join of F.sub.1 with itself. Accordingly, the pass 2 can be optimized by replacing the join with C.sub.2 by a join with F.sub.1 before the table function (see FIG. 10). That way, the table function gets only frequent items and generates significantly fewer 2-item combinations. This optimization can be useful for other passes too, but unlike for pass 2, they still require a join with C.sub.k. FIG. 11 illustrates the support counting using GatherJoin in the second pass on the data. The pseudo-code for a typical implementation of the second-pass optimization is as follows.

Detailed Description Text (112):
from (select item.sub.1, item.sub.2, CountIntersect(t.sub.1.tid-list, t.sub.2.tid-list) as cnt

Detailed Description Text (117):
GatherCount: One variation of the GatherJoin approach for pass two is the GatherCount approach where the group-by is performed inside the table function instead of doing it outside. The candidate 2-itemsets (C.sub.2) are represented as a two dimensional array inside the modified table function Gather-Cnt for doing the support counting. Instead of outputting the 2-item combinations of a transaction, it directly uses it to update support counts in the memory and output only the frequent 2-itemsets, F.sub.2 and their support after the last transaction. Thus, the table function Gather-Cnt is an extension of the GatherComb-2 table function used in GatherJoin. The absence of the outer grouping makes this option an attractive solution. The UDF code is also small since it only needs to maintain a 2-dimension array. It can be applied for subsequent passes but for the added complexity of maintaining hash-tables to index the C.sub.k s itemset. The disadvantage of this approach is that it can require a large amount of memory to store C.sub.2. If enough memory is not available, C.sub.2 needs to be partitioned and the process has to be repeated for each partition.

Detailed Description Text (118):
Gather Prune: A potential disadvantage of GatherJoin is the high cost of joining the large number of item combinations with C.sub.k. The join with C.sub.k can be done in the table function to reduce the number of such combinations. C.sub.k is converted to a BLOB and passed as an argument to the table function. The cost of passing the BLOB for every tuple of R could be high. However, the parameter passing cost can be reduced by using a smaller BLOB that only approximates the real C.sub.k. The trade-off is increased cost for other parts notably grouping because not as many combinations are filtered.

Detailed Description Text (120):
The choice of the best SQL-OR approach depends on a number of data characteristics like the number of items, total number of transactions, average length of a transaction etc. The costs of different approaches in each pass can be determined in terms of parameters that are known or can be estimated after the candidate generation step of each pass, as defined in Table 1 below. The cost of GatherJoin includes the cost of generating k-item combinations, joining with C.sub.k, and grouping to count the support. The number of k-item combinations generated is C(N, k)*T. The join with C.sub.k filters out the non-candidate item combinations. The size of the join result is the sum of the support of all the candidates denoted by

Detailed Description Text (121):
S(C.sub.k). The actual value of the support of a candidate itemset will be known only after the support counting phase. The total cost of the GatherJoin approach is found as:

Detailed Description Text (123):
This cost needs to be modified to reflect the special optimization of joining with F1 to consider only frequent items. For the second pass, the total cost of GatherJoin is: ##EQU4##

Detailed Description Text (129):
where Blob(k*C.sub.k) denotes the BLOB passing cost since each itemset in C.sub.k contains k items.

Detailed Description Text (130):
The cost estimate of the Horizontal method is similar to that of GatherJoin except that here the data is materialized in the horizontal format before generating the item combinations.

Detailed Description Text (132):
In this approach, the data table is first transformed into a vertical format by creating for each item a BLOB containing all tids that contain that item (Tid-list creation phase) and then count the support of itemsets by merging together these tid-lists (support counting phase).

Detailed Description Text (133):
A table function Gather is used for creating the Tid-lists. This is the same as the Gather function in GatherJoin except here, the tid-list is created for each frequent item. The data table T is scanned in the (item, tid) order and passed to the function Gather. The function collects the tids of all tuples of T with the same item in memory and outputs a (item, tid-list) tuple for items that meet the minimum support criterion. The tid-lists are represented as BLOBs and stored in a new TidTable with attributes (item, tid-list).

Detailed Description Text (134):
In the support counting phase, conceptually for each itemset in C.sub.k the tid-lists of all k items are collected and the number of tids in the intersection of these k lists is counted using a user defined function (UDF). The tids are in the same sorted order in all the tid-lists and therefore the intersection can be done easily and efficiently by a single pass of the k lists. This conceptual step can be improved further by decomposing the intersect operation so that we can share these operations across itemsets having common prefixes as follows:

Detailed Description Text (135):
First, distinct pairs of items (item.sub.1, item.sub.2) are selected from C.sub.k. For each distinct pair, an intersect operation is performed to get a new result-tidlist. Distinct triples (item.sub.1, item.sub.2,item.sub.3) are then determined from C.sub.k with the same first two items. The result-tidlist is intersected with tid-list for item3 for each triple, then with item.sub.4 and so on until all k tid-lists per itemset are intersected.

Detailed Description Text (137):
insert into F.sub.k select item.sub.1, . . . , item.sub.k, count(tid-list) as cnt

Detailed Description Text (140):
select item.sub.1, . . . , item.sub.1,

Detailed Description Text (143):
(select distinct item.sub.1, . . . item.sub.1 from C.sub.k) as d.sub.1

Detailed Description Text (149):
from (select item.sub.1, item.sub.2, Count_Intersect(t1.tid-list, t2.tid-list) as cnt

Detailed Description Text (153):
The cost of the Vertical approach during support counting phase is dominated by the cost of invoking the UDFs and intersecting the tid-lists. The UDF is first called for each distinct

item pair in C.sub.k, then for each distinct item triple with the same first two items and so on. Let d.sub.j.sup.k be the number of distinct j item tuples in C.sub.k. Then the total number of times the UDF is invoked is ##EQU5##

Detailed Description Text (158):
The performance results for mining boolean association rules using the above methods are now described. The results for mining generalized association rules and sequential patterns are similar. All of the experiments were performed on Version 5 of IBM DB2 Universal Server installed on a RS/6000 Model 140 with a 200 MHz CPU, 256 MB main memory and a 9 GB disc with a measured transfer rate of 8 MB/s. A collection of four real-life datasets were used. These datasets have differing values of parameters like the total number of (tid,item) pairs, the number of transactions (tids), the number of items and the average number of items per transaction. Table 2 summarizes these parameters.

Detailed Description Text (159):
A composite index (item.sub.1, . . . , item.sub.k) is built on C.sub.k, k different indices on each of the k items of C.sub.k and a (tid, item) and a (item, tid) index on the data table. The index building cost is not reflected in the total time.

Detailed Description Text (161):
FIGS. 13A-13D show the performance of the four SQL-OR approaches: GatherJoin method and its variants GatherPrune, GatherCount, and Vertical, on four datasets for different support values. Overall, the Vertical approach has the best performance and is sometimes more than an order of magnitude better than the other three approaches. The majority of the time of the Vertical approach is spent in transforming the data to the Vertical format in most. The vertical representation is like an index on the item attribute. Its performance is even better if this time is viewed as a one-time activity like index building.

Detailed Description Text (162):
The time taken is broken down by each pass and an initial "prep" stage where any one-time data transformation cost is included. Note that the time to transform the data to the Vertical format was much smaller than the time for the horizontal format although both formats write almost the same amount of data. The main reason was the difference in the number of records written. The number of frequent items is often two to three orders of magnitude smaller than the number of transactions.

Detailed Description Text (163):
Between GatherJoin and GatherPrune, neither strictly dominates the other. The special optimization in GatherJoin of pruning based on F1 had a big impact on performance. When these different approaches are compared based on the time spent in each pass, no single approach is "the best" for all different passes of the different datasets especially for the second pass. For pass three onwards, Vertical is often two or more orders of magnitude better than the other approaches. Two factors that affect the choice amongst the Vertical, GatherJoin and GatherCount approaches in different passes and pass 2 in particular are: number of frequent items (F1) and the average number of frequent items per transaction (Nf).

Detailed Description Text (165):
Impact of longer names: In this study, it is assumed that the tids and item ids are all integers. Often in practice these are character strings longer than four characters. The two (already expensive) steps that could suffer because of longer names are (1) final group-bys during pass 2 or higher when the GatherJoin approach is chosen and (2) tid-list operations when the Vertical approach is chosen. For efficient performance, the first step requires a mapping of item ids and the second one requires us to map tids. A table function is used to map the tids to unique integers efficiently in one pass and without making extra copies. The input to the table function is the data table in the tid order. The table function remembers the previous tid and the maintains a counter. Every time the tid changes, the counter is incremented. This counter value is the mapping assigned to each tid. The tid mapping is only needed to be done once before creating the TidTable in the Vertical approach. These two steps therefore can be pipelined. The item mapping is done slightly differently. After the first pass, a column is added to F1 containing a unique integer for each item, and similarly for the

TidTable. The GatherJoin approach already joins the data table T with F1 before passing to table function Gather. Therefore, the integer mappings of each item can be passed to GatherJoin from F1 instead of its original character representation. After these two transformations, the tid and item fields are integers for all the remaining queries including candidate generation and rule generation. By mapping the fields this way, the longer names are expected to have similar performance impact on all of the described options.

Detailed Description Text (167):
Various preferred embodiments for an integrated mining and database system have been described, using association rule mining as an example. Some of the embodiments are implemented using SQL-92 while others are constructed using new object-relational extensions like UDFs, BLOBs, Table functions etc. The integrated systems built with the extended SQL provide orders of magnitude improvement over the SQL-92 based-implementations.

Detailed Description Paragraph Table (1):
TABLE 1 Notation used for cost analysis of different approaches R number of records in the input transaction table T number of transactions N average number of items per transaction ##EQU1## F.sub.1 number of frequent items S(C) sum of support of each itemset in set C R.sub.f number of records out of R involving frequent items = S(F.sub.1) N.sub.f average number of frequent items per transaction ##EQU2## C.sub.k number of candidate k-itemsets C (N, k) number of combinations of size k possible out of a set of size n: ##EQU3## s.sub.k cost of generating a k item combination using table function Comb-k group (n, m) cost of grouping n records out of which m are distinct join (n, m, r) cost of joining two relations of size n and m to get a result of size r blob (n) cost of passing a BLOB of size n integers as an argument

Detailed Description Paragraph Table (2):
TABLE 2 Comparison of different SQL approaches # Records # Transactions # Items Avg.# items in millions in millions in thousands per transaction Datasets (R) (T) (I) (R/T) Dataset-A 2.5 0.57 85 4.4 Dataset-B 7.5 2.5 15.8 2.62 Dataset-C 6.6 0.21 15.8 31 Dataset-D 14 1.44 480 9.62

Other Reference Publication (1):
Tang et al., "Using Incremental Pruning to Increase the Efficiency of Dynamic Itemset Counting for Mining Asssociation Rules", ACM 1998.*

Other Reference Publication (7):
R. Agrawal et al., "Mining Association Rules between Sets of Items in Large Databases," Proceedings of the ACM-SIGMOD 1993 Int'l Conference on the Management of Data, Washington, DC, 1993, pp. 207-216.

Other Reference Publication (10):
S. Brin et al., "Dynamic Itemset Counting and Implication Rules for Market Basket Data," Proceedings of the ACM SIGMOD Conference on Management of Data, 1997.

Other Reference Publication (22):
R. Srikant et al., "Mining Association Rules with Item Constraints," Proceedings of the 3rd Int'l Conference on Knowledge Discovery in Databases and Data Mining, Oregon, 1996, pp. 67-73.

CLAIMS:

1. A method for mining rules from an integrated database and data-mining system having a table of data transactions and a query engine, the method comprising the steps of:

a) performing a group-by query on the transaction table to generate a set of frequent 1-itemsets;

b) determining frequent 2-itemsets from the frequent 1-itemsets and the transaction table;

c) generating a candidate set of (n+2)-itemsets from the frequent (n+1)-itemsets, where n=1;

d) determining frequent (n+2)-itemsets from the candidate set of (n+2)-itemsets and the

transaction table using a query operation;

e) repeating steps (c) and (d) with n=n+1 until the candidate set is empty; and

f) generating rules from the union of the determined frequent itemsets.

2. The method as recited in claim 1, wherein:

each transaction corresponds to a plurality of items; and

the step of performing a group-by query includes the steps of counting the number of transactions that contain each item and selecting the items that have a support above a user-specified threshold in determining the frequent 1-itemsets, the support of an item being the number of transactions that contain the item.

3. The method as recited in claim 1, wherein the frequent 2-itemsets are generated using a join query.

4. The method as recited in claim 3, wherein the step of determining frequent 2-itemsets includes the steps of:

joining an 1-itemset with itself and two copies of the transaction table using join predicates;

grouping results of the joining step for a pair of items to count the support of the items in the pair; and

removing all 2-itemsets having a support below a specified threshold.

5. The method as recited in claim 1, wherein the candidate (n+2)-itemsets are generated using an (n+2)-way join operation.

6. The method as recited in claim 1, wherein the frequent (n+2)-itemsets are determined using an (n+3)-way join query, on the candidate itemsets and (n+2) copies of the transaction table.

7. The method as recited in claim 1, wherein:

the frequent (n+2)-itemsets are determined using cascaded subqueries; and

the step of determining frequent (n+2)-itemsets includes the steps of:

selecting distinct first items in the candidate itemsets using a subquery;

joining the distinct first items with the transaction table;

cascading (n+1) subqueries where each j-th subquery is a three-way join of the result of the last subquery, distinct j items from the candidate itemsets, and the transaction table; and

determining which of the (n+2)-itemsets are frequent using the results of the last subqueries.

8. The method as recited in claim 1, wherein the step of generating rules includes the steps of:

putting all items from the frequent itemsets into a table F;

generating a set of candidate rules from the table F using a table function;

joining the candidate rules with the table F; and

filtering out from the candidate rules those that do not meet a confidence criteria.

9. The method as recited in claim 1, wherein:

the query engine is an object-relational engine; and

the steps of generating 2-itemsets and (n+1)-itemsets include the steps of:

selecting from Vertical, GatherCount, and GatherJoin a method having the best cost for generating the (n+1)-itemsets;

if the Vertical method is selected, then transforming the transaction table into a vertical format; and

generating the frequent (n+1)-itemsets using the selected method.

10. The method as recited in claim 9 wherein the cost for generating the itemsets is based on statistics about the data transactions and the candidate itemsets.

12. An integrated data-mining system comprising:

a database having a table of data transactions;

a query engine coupled to the database;

a query preprocessor for generating queries and input parameters for the query engine;

means for performing a group-by query on the transaction table to generate a set of frequent 1-itemsets;

means for determining frequent 2-itemsets from the frequent 1-itemsets and the transaction table;

means for generating a candidate set of (n+2)-itemsets from the frequent (n+1)-itemsets, where n=1;

means for determining frequent (n+2)-itemsets from the candidate set of (n+2)-itemsets and the transaction table using a query having cascaded subqueries;

means for repeatedly generating the candidate set of (n+2)-itemsets and determining frequent (n+2)-itemsets with n=n+1 until the candidate set is empty; and

means for generating rules from the union of the determined frequent itemsets.

13. A computer program product for mining rules from an integrated database and data-mining system having a table of data transactions and a query engine, the product comprising:

a) means, recorded on the recording medium, for directing the system to perform a group-by query on the transaction table to generate a set of frequent 1-itemsets;

b) means, recorded on the recording medium, for directing the system to determine frequent 2-itemsets from the frequent 1-itemsets and the transaction table;

c) means, recorded on the recording medium, for directing the system to generate a candidate set of (n+2)-itemsets from the frequent (n+1)-itemsets, where n=1;

d) means, recorded on the recording medium, for directing the system to determine frequent (n+2)-itemsets from the candidate set of (n+2)-itemsets and the transaction table using a query operation;

e) means, recorded on the recording medium, for directing the system to repeatedly generate the

candidate set of (n+2)-itemsets and determine frequent (n+2)-itemsets with n=n+1 until the candidate set is empty; and

f) means, recorded on the recording medium, for directing the system to generate rules from the union of the determined frequent itemsets.

14. The data-mining system as recited in claim 12, wherein:

each transaction corresponds to a plurality of items; and

the means for performing a group-by query includes means for counting the number of transactions that contain each item and selecting the items that have a support above a user-specified threshold in determining the frequent 1-itemsets, the support of an item being the number of transactions that contain the item.

15. The data-mining system as recited in claim 12, wherein the frequent 2-itemsets are generated using a join query.

16. The data-mining system as recited in claim 15, wherein the means for determining frequent 2-itemsets includes:

means for joining an 1-itemset with itself and two copies of the transaction table using join predicates;

means for grouping results of the joining step for a pair of items to count the support of the items in the pair; and

means for removing all 2-itemsets having a support below a specified threshold.

17. The data-mining system as recited in claim 12, wherein the candidate (n+2)-itemsets are generated using an (n+2)-way join operation.

18. The data-mining system as recited in claim 12, wherein the frequent (n+2)-itemsets are determined using an (n+3)-way join query, on the candidate itemsets and (n+2) copies of the transaction table.

19. The data-mining system as recited in claim 12, wherein:

the frequent (n+2)-itemsets are determined using cascaded subqueries; and

the means for determining frequent (n+2)-itemsets includes:

means for selecting distinct first items in the candidate itemsets using a subquery;

means for joining the distinct first items with the transaction table;

means for cascading (n+1) subqueries where each j-th subquery is a three-way join of the result of the last subquery, distinct j items from the candidate itemsets, and the transaction table; and

means for determining which of the (n+2)-itemsets are frequent using the results of the last subqueries.

20. The data-mining system as recited in claim 12, wherein the means for generating rules includes:

means for putting all items from the frequent itemsets into a table F;

means for generating a set of candidate rules from the table F using a table function;

means for joining the candidate rules with the table F; and

means for filtering out from the candidate rules those that do not meet a confidence criteria.

21. The data-mining system as recited in claim 12, wherein:

the query engine is an object-relational engine; and

the means for generating 2-itemsets and (n+1)-itemsets include:

means for selecting from Vertical, GatherCount, and GatherJoin a method having the best cost for generating the (n+1)-itemsets;

if the Vertical method is selected, then means for transforming the transaction table into a vertical format; and

means for generating the frequent (n+1)-itemsets using the selected method.

22. The data-mining system as recited in claim 21, wherein the cost for generating the itemsets is based on statistics about the data transactions and the candidate itemsets.

24. The product as recited in claim 13, wherein:

each transaction corresponds to a plurality of items; and

the means for directing to perform a group-by query includes means, recorded on the recording medium, for directing the system to count the number of transactions that contain each item and selecting the items that have a support above a user-specified threshold in determining the frequent 1-itemsets, the support of an item being the number of transactions that contain the item.

25. The product as recited in claim 13, wherein the frequent 2-itemsets are generated using a join query.

26. The product as recited in claim 25, wherein the means for directing to determine frequent 2-itemsets includes:

means, recorded on the recording medium, for directing the system to join an 1-itemset with itself and two copies of the transaction table using join predicates;

means, recorded on the recording medium, for directing the system to group results of the joining step for a pair of items to count the support of the items in the pair; and

means, recorded on the recording medium, for directing the system to remove all 2-itemsets having a support below a specified threshold.

27. The product as recited in claim 13, wherein the candidate (n+2)-itemsets are generated using an (n+2)-way join operation.

28. The product as recited in claim 13, wherein the frequent (n+2)-itemsets are determined using an (n+3)-way join query, on the candidate itemsets and (n+2) copies of the transaction table.

29. The product as recited in claim 13, wherein:

the frequent (n+2)-itemsets are determined using cascaded subqueries; and

the means for directing to determine frequent (n+2)-itemsets includes:

means, recorded on the recording medium, for directing the system to select distinct first

items in the candidate itemsets using a subquery;

means, recorded on the recording medium, for directing the system to join the distinct first items with the transaction table;

means, recorded on the recording medium, for directing the system to cascade (n+1) subqueries where each j-th subquery is a three-way join of the result of the last subquery, distinct j items from the candidate itemsets, and the transaction table; and

means, recorded on the recording medium, for directing the system to determine which of the (n+2)-itemsets are frequent using the results of the last subqueries.

30. The product as recited in claim 13, wherein the means for directing to generate rules includes:

means, recorded on the recording medium, for directing the system to put all items from the frequent itemsets into a table F;

means, recorded on the recording medium, for directing the system to generate a set of candidate rules from the table F using a table function;

means, recorded on the recording medium, for directing the system to join the candidate rules with the table F; and

means, recorded on the recording medium, for directing the system to filter out from the candidate rules those that do not meet a confidence criteria.

31. The product as recited in claim 13, wherein:

the query engine is an object-relational engine; and

the means for directing the system to generate 2-itemsets and (n+1)-itemsets include:

means, recorded on the recording medium, for directing the system to select from Vertical, GatherCount, and GatherJoin a method having the best cost for generating the (n+1)-itemsets;

if the Vertical method is selected, then means, recorded on the recording medium, for directing the system to transform the transaction table into a vertical format; and

means, recorded on the recording medium, for directing the system to generate the frequent (n+1)-itemsets using the selected method.

32. The product as recited in claim 31, wherein the cost for generating the itemsets is based on statistics about the data transactions and the candidate itemsets.

Previous Doc Next Doc Go to Doc#

Freeform Search

Database:
- US Pre-Grant Publication Full-Text Database
- **US Patents Full-Text Database**
- US OCR Full-Text Database
- EPO Abstracts Database
- JPO Abstracts Database
- Derwent World Patents Index
- IBM Technical Disclosure Bulletins

Term: L13 and item$

Display: 100 **Documents in Display Format:** FRO **Starting with Number** 1

Generate: ○ **Hit List** ◉ **Hit Count** ○ **Side by Side** ○ **Image**

Search Clear Interrupt

Search History

DATE: Friday, September 29, 2006 **Purge Queries** Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=OR			
L14	L13 and item$	1	L14
L13	L12 and fields	3	L13
L12	L11 and relationship	3	L12
L11	L10 and type	8	L11
L10	L9 and instance	8	L10
L9	L8 and schemas	8	L9
L8	object near relational near extension	8	L8
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L7	L6 and (instance near type)	16	L7
L6	L5 and relationship	16	L6
L5	L4 and item$	16	L5
L4	L3 and field$	16	L4
L3	L2 and instance	17	L3
L2	L1 and (extend$ near schema$)	17	L2
L1	object near relational near extension	56	L1

END OF SEARCH HISTORY